USE OF A COMMERCIAL VISUAL PROGRAMMING LANGUAGE TO SIMULATE, DECOMMUTATE, TEST AND DISPLAY A TELEMETRY STREAM

George Wells and Ed Baroth, Ph.D.
Measurement Technology Center,
Jet Propulsion Laboratory,
California Institute of Technology

ABSTRACT

The advantages of using visual programming to create, modify, test and display a telemetry stream are presented. 'I'he failure to fully deploy the high-gain antenna of the Galileo spacecraft has resulted in a software redesign of the computer systems onboard the spacecraft to support the low-gain antenna mission. Visual programming software is being used to test new algorithms as part of the ground support for the spacecraft Test Bed. It is very important that any new software algorithms be thoroughly tested on the ground before any modifications are made to the spacecraft.

The advantage of using a visual programming language (LabVIEW, National Instruments) is that it provides easy visibility into the decommutation process that is being modified by the Galileo programming support team. In addition, utilities were written using visual programming to allow real-time data display and error detection. A data acquisition board is used to clock in the actual synchronous telemetry signal from the Test Bed at rates below 10 kHz. The time to write and modify the code using visual programming is significantly less (by a factor of 4 to 10) than using text-based code. The gains in productivity are attributed to the communication among the customer, developer, and computer that are facilitated by the visual syntax of the language,

KEY WORDS

Visual programming language, telemetry simulation and decommutation, LabVIEW, software productivity comparisons.

INTRODUCTION

The Measurement Technology Center (MTC) evaluates commercial data acquisition, analysis, display and control hardware and software products that are then made available to experimenters at the Jet Propulsion Laboratory. In addition, the MTC specifically configures and delivers turn-key measurement systems that include software., user interface, sensors (e.g., thermocouples, pressure transducers) and signal conditioning, plus data acquisition, analysis, display, simulation and control capabilities."?

Visual programming tools are frequently used to simplify development (compared to text-based programming) of such systems. Employment of visual programming tools that control off-the-shelf interface cards has been the most important factor in reducing time and cost of configuring these systems. The MTC consistently achieves a reduction in software/system development time by at least a factor of four, and up to an order of magnitude, compared to text-basecl software tools.[3,4,5,6] Others in industry are reporting similar increases in productivity and reduction in software/system development time and cost.[7,8,9]

BACKGROUND

The Galileo spacecraft is scheduled for an encounter with Jupiter in December of 1995. About six months before encounter it will release a probe that will impact Jupiter. The spacecraft will then change its trajectory to go into a highly elliptical orbit with a period of three months. A timer in the probe will activate its radio transmitter just before arriving at Jupiter. The original plan called for a real-time relay of the probe radio signal by the Galileo spacecraft to earth, but this is no longer possible because its high-rate dish antenna has failed to open fully. Instead, the Galileo computer will be re-programmed to strip out the overhead and house-keeping bits in the data stream coming from the probe. The important data sent by the probe transmitter will be stored in the limited on-board memory for later down-linking to earth. This down-linking will occur sometime during the first orbit using the low-gain omni-directional antenna, at a much lower bit rate than if the high gain antenna was fully functional.

Currently, the MTC is supporting a software redesign of the computer system aboard the Galileo spacecraft. This paper documents the programming effort to verify the correct re-programming of the Galileo computer subsystems by monitoring the telemetry of the ground Test Bed setup of the computer subsystems and the emulation hardware for the probe radio receivers to assure that every byte is correctly downloaded. The MTC is using LabVIEW software among other tools to help test the flight software redesign. For details on the LabVIEW environment, other sources exist. [10,11,12]

The task of the computer redesign is complicated by the fact that there are actually two redundant probe radio receivers and several multiple computer subsystems with their own memory partitions on the orbiter. A Configuration Table was set up to specify which receiver(s) would be the source of the data and which sections of which subsystems would be the destinations of the stripped data.

Using visual programming, software was developed to perform a stripping algorithm on the emulated probe data to be used in the test, and, using the Configuration Table, create a Predict Table of the data to be downloaded (Figure 1). A probe analyzer program was developed to monitor the telemetry from the Test Bed, decommutate the memory read-out data, compare it to the Predict Table and display the progress of the test. After the test was over, utilities developed in LabVIEW were used to disposition any discrepancies, including incorrect or missing bytes, and relate them to the original expanded data to determine the nature of the problem. A neccssary additional component was a Test Bed simulator so all of the other programs could-be developed and debugged before connection to the Test Bed.

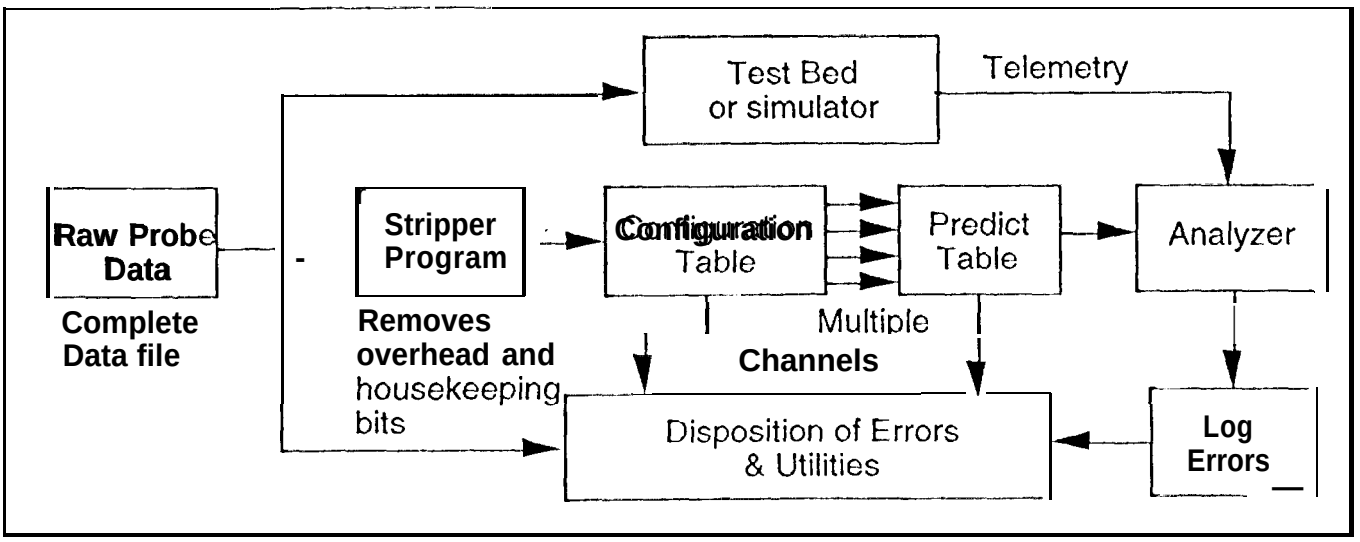

Figure 1. Schematic of ground support sequence of data flow from probe.

LabVIEW running on a Macintosh Quadra was used as the programming environment for this task because it had proved to be superior in similar tasks.[13] The advantages LabVIEW provides include the ease with which the customer can communicate requirements to the programmers and understand the operation of the program so that changes can be suggested. The gains in productivity are attributed to the communication among the customer, developer, and computer that are facilitated by the visual syntax of the language. LabVIEW proved exceptionally capable in providing an integrated environment to manage all aspects of the telemetry test, from pre-test data set-up to post-test discrepancy resolution, as well as running the test in several simulator modes or with the Test Bed.

PREDICT TABLE GI1NERATION

The creation of the Predict Table was performed in two steps. First, files containing the simulated raw data from the two radio receivers had to be stripped using the same algorithm to be programmed into the Galileo computers. The fact that the processors are different is considered to be a check on the programmers' understandings of the algorithm. Second, the Configuration Table was used to determine where the stripped data would be stored in the on-board memory of the Test Bed and in the Predict Table of the Analyzer. The Stripper program, Configuration Table, and Predict Table are LabVIEW emulations of the identical operations performed in the Test Bed, except instead of storing the bytes in various subsystem memories, as they are in the Test Bed, they are saved to the Predict Table disk file.

TELEMETRY SIMULATOR PROGRAM

The telemetry used to download the Galileo probe data makes use of the memory readout mode that is an available option in every frame. The first one hundred bytes of each frame type have identical meanings; the remaining bytes differ depending on frame type, but are of no concern for this test. This is shown as Figure 2. Included in the first one hundred bytes are four bytes for a psuedo-random sync code (Sync), two bytes comprising frame ID (FID) which included a bit to signify when the frame is in the memory read-out mode, six bytes of spacecraft clock (SCLK), five house-keeping bytes, one byte to signify which computer subsystem is being read out (S/S), two bytes pointing to the first address being read out (ADD), and eighty bytes of memory read-out.

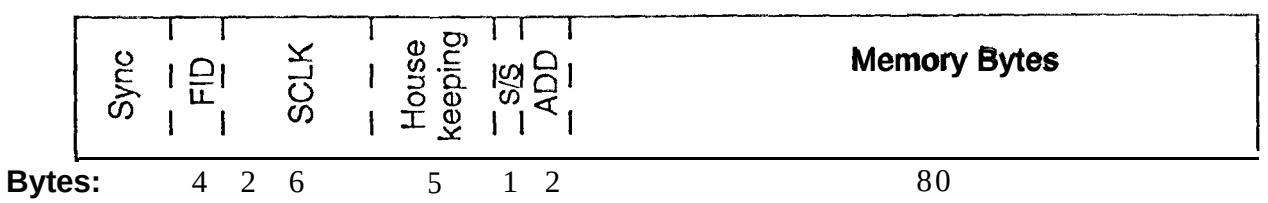

Figure 2. Memory map of the first 100 bytes of each frame type.

Once the operator has started the memory read-out mode, the probe simulator program will randomly select one of the computer subsystems as defined by the Configuration Table and output the next eighty bytes as defined by the Predict Table, When all the bytes for a subsystem have been read out, the memory read-out bit is turned off for that frame unless the operator has put the simulator in recirculate mode in which case that subsystem will be downloaded another time.

If the test mode requires that the simulator actually generate the telemetry, the two digital-to-analog channels of the multifunction input-output (MIO) board are initialized to generate a pair of continuous signals; one of them is a square wave of the frequency specified by the operator ant{ the other one is the

data. Both have zero to five-volt swings. If the test mode does not require real telemetry. then the data bit-stream is saved in memory. The simulator is used for debugging or demonstration purposes only.

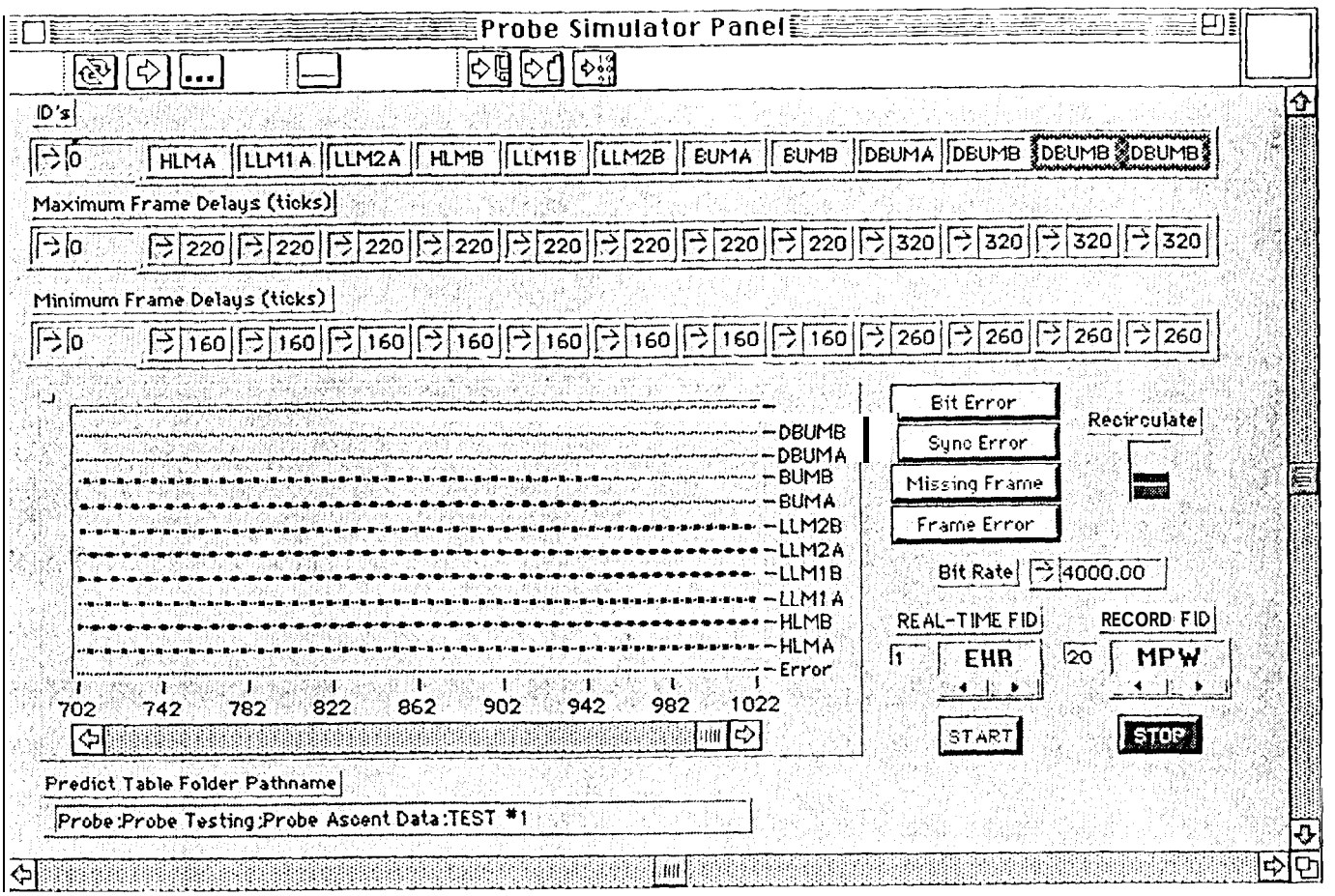

Figure 3. Telemetry simulator user interface (LabVIEW VI Front Panel)

Figure 3 is the user interface (or VI for Virtual instrument Front Panel) for the simulator program. It displays the names of the subsystems along the top and the rates at which frames arc generated. The user can also generate errors of various types. The subsystem type for each frame is displayed as a mark on a scrolling strip chart.

Figure 4 shows the program (LabVIEW Diagram) associated with the simulator (Figure 3). It has an icon for each of the controls on the Panel and gives a general idea of how data flows between the various icons, structures, and subroutines (subVI's). An example of a subVI is the Predict Table box near the middle of the figure. It can be expanded to show its own user interface and program. The large rectangular frames arc LabVIEW structures that help organize the diagram and perform looping functions. Details of LabVIEW programming can be found elsewhere.

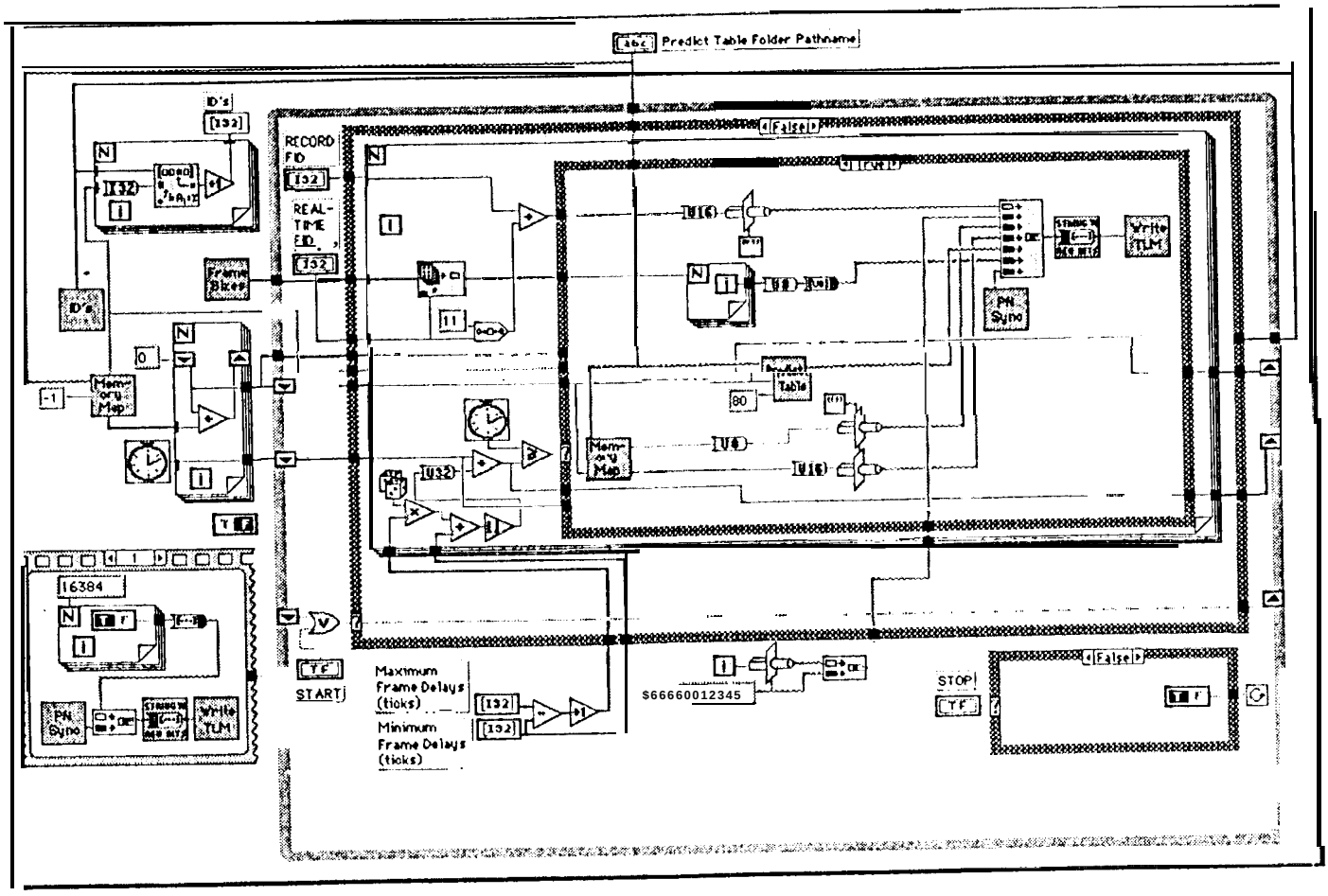

Figure 4. Telemetry simulator program (LabVIEW VI Diagram).

TELEMETRY ANALYZER PROGRAM

If the test mode requires the analyzer program to read the real telemetry signal, the MIO board is initialized to take continuous readings on a single analog channel clocked in by the clock signal. If the test mode does not require real telemetry, then the requested bits come from the memory saved by the simulator. In either case, the analyzer program will wait until the requested bits are available.

During initial development, the simulator and analyzer communicated directly through a common memory software routine on the same Mac computer. Later, the simulator generated a real clock and data stream on one Mac that the analyzer read on another Mac using National Instruments A/D boards. When it came time to connect the analyzer to the lest Bed telemetry stream, it was operating within a few hours.

During the course of simulated or actual monitoring of the telemetry signal, the memory read-out contents are compared with locations in the Predict Table as specified by the Configuration Table. As long as the contents match, only the subsystem name and address are saved, but in the event of an error, the entire eighty bytes for both the Predict Table and the telemetry stream are saved. Also, a count of the number of times each byte in the Predict Table is read is maintained, along with a flag indicating if any byte was read in error. A counter displays the remaining number of bytes from the Predict Table that

A bit-map displays a pixel for each byte in the Predict Table and turns black if it has been read correctly. Thus, it is very easy to monitor the progress of the test. have not yet been readout.

Figure 5 shows the analyzer in operation and generally reflects the same subsystem patterns as the simulator except delayed in time. The "Predict Tables" and "Downloaded" windows will show the actual hex bytes whenever there is a discrepancy. Other Indicators display various errors or status.

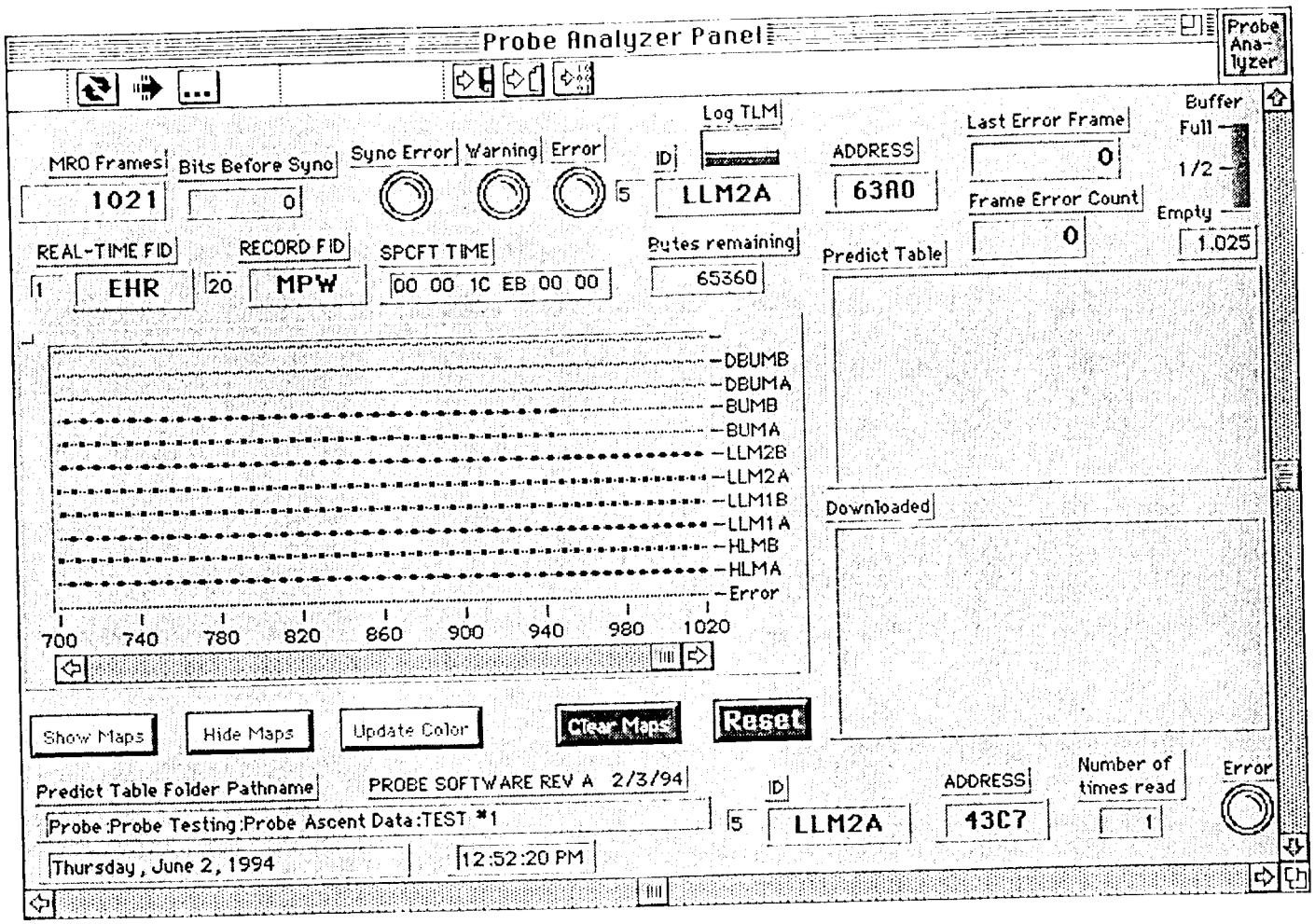

Figure 5. Telemetry analyzer user interface (LabVIEW VI Front Panel).

DISPOSITION OF DISCREPANCIES

To better analyze any errors that may occur during a test, additional utilities were developed to pinpoint the location of the error in the Predict Table and trace it back to the corresponding error bits in the original raw data files. One of these utilities would display the raw and the stripped data as defined by the Predict Table and as defined by the Galileo telemetry so that the system and nature of the source of the errors within any reasonable time. Typically, however, tests of this nature have been run in the past without such utilities, simply because of the difficulty and cost of developing them and because of the optimistic hope that they would never be needed. be identified. Without these utilities, it would have been virtually impossible to determine

ADDITIONAL TASKS

Since LabVIEW promotes high programmer productivity, the customer had plenty of time to suggest new features such as the ability to log the raw telemetry stream to a disk file and then simulate the telemetry stream from disk so that a test could be repeated without incurring the cost of re-running the actual test using the Test Bed and support personnel. This was a simple modification of the original implementation using the simulator and analyzer on the same Mac and proved to be valuable because, as it turned out, during the actual first run of the Test Bed a wrong Configuration Table was used which generated a large number of errors. When re-run from the logged disk file with the correct Configuration Table, there were no errors, saving a costly re-run using the Test Bed personnel.

Another suggestion that was implemented was the ability of the analyzer to "read" telemetry from the TCP/IP port instead of a disk file. This allowed communication between the analyzer and a Test Bed emulator that was written on a Sun workstation. The Sun provided the Galileo programmers with an easier and faster platform on which to develop their code. (The Galileo CPU's are ancient 1802 8-bit processors.)

EXAMPLE OF VISUAL/ TEXT-BASED PROGRAMMING COMPARISON

During the course of program development, it became necessary to generate the source files for the probe receivers based on data that had previously been downloaded from the Galileo spacecraft telemetry in a test mode. A "C" programmer who was familiar with the file structure was assigned this task for approximately two weeks and had not delivered a finished product. As a back-up, the task was also given to a LabVIEW programmer. The application was developed in LabVIEW by one programmer in less than 2 weeks. For the one data file which the routine was written to operate on, no errors were found in the input data.

After spending approximately one month on the task, the C programmer delivered a product that was advertised as being complete. The C. programmer noted that when the program ran, it found errors in the input data. Since the customer already had the LabVIEW version (with no errors), the raw data was compared with the two sets of output files and it was discovered that a bug in the C code caused it to find bad data where none existed. The C programmer spent a week or so making changes to the code, but was still not able to process the data correctly. The customer then decided to halt the C programming effort.

CONCLLJS1ONS

A visual programming language was able to create, modify, test and display a telemetry stream. It provided easy visibility into the decommutation process modified by the Galileo programming support team. The time to write and modify the code using visual programming was significantly less (by a factor of 4 to 10) than using text-based code. This task showed that it is possible to use visual programming for realistic programming applications. It also confirmed that visual programming can significantly reduce software development time compared to text-based programming.

Other advantages demonstrated were in the areas of prototyping and verification. Different approaches, can be demonstrated and evaluated quickly using a visual programming language. Verification can be demonstrated using the graphical user interface features available in a visual programming language easier than using conventional text-based code.

As stated, the gains in productivity arc attributed to the communication among the customer, developer, and computer that are facilitated by the visual syntax of the language. The advantages LabVIEW

provides include the ease with which the customer can communicate requirements to the programmers and understand the operation of the program so that changes can be suggested. With this communication, the boundaries between requirements, design, development, and test appear to collapse.

ACKNOWLEDGMENTS

The research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

The authors wish to acknowledge the contributions of Amy Walsh and Lee Johnsen towards the writing of this paper.

REFERENCES

1. Baroth, E. C., Clark, D. J. and Losey, R. W., "Acquisition, Analysis, Control, and Visualization of Data Using Personal Computers and a Graphical-Based Programming Language," Session 2659, Conference Proceedings of American Society of Engineering Educators (ASEE), Toledo, Ohio, June 21-25, 1992, pp. 1447-1453.
2. Baroth, E. C., Clark, D. J. and Losey, R. W., "An Adaptive Structure Data Acquisition System using a Graphical-Based Programming Language," AIAA-92-4833-CP, Conference Proceedings of Fourth AIAA/Air Force/NASA/OAI Symposium on Multidisciplinary Analysis and Optimization, Cleveland, Ohio, September 21-23, 1992, pp. 1104-1110.
3. Baroth, E. C., Hartsough, C., Johnsen, L., McGregor, J., Powell-Meeks, M., Walsh, A., Wells, G., Chazanoff, S., and Brunzie, T. "A Survey of Data Acquisition and Analysis Software Tools, Part 1," Evaluation Engineering Magazine, October, 1993, pp. 54-66.
4. Breeman, D., "Jet Propulsion Lab Aids in Space Craft Project," Scientific Computing and Automation, November, 1993, pp. 26-28.
5. Bulkeley, D., "Today's Equipment Tests Tomorrow's Designs," Desire News Magazine, May 17, 1993, pp. 82-86.
6. Puttre', M., "Software Makes Its Home in the Lab," Mechanical Engineering Magazine, October, 1992, pp. 75-78.
7. Kent, G., "Automated RF Test System for Digital Cellular Telephones," Proceedings from NEPCON West '93, Anaheim, California, February 7-11, 1993, pp. 1055-1064.
8. Henderson, J. R., "Sequential File Creation for Automated Test procedures," Proceedings from NEPCON West '93, Anaheim, California, February 7-11, 1993, pp. 1065-1077.
9. Jordan, S. C., "Cutting Costs the Old Fashioned Way," Proceedings from NEPCON West '93, Anaheim, California, February 7-11, 1993, pp. 1921-1931.
10. National instruments Catalog, 1994, pp. 17-112.
11. Baroth, E. C., Hartsough, C., Johnsen, L., McGregor, J., Powell-Meeks, M., Walsh, A., Wells, G., Chazanoff, S., and Brunzie, T., "A Survey of Data Acquisition and Analysis Software Tools, Part 2," Evaluation Engineering Magazine, November, 1993, pp. 128-140.
12. Small, C. H., "Diagram Compilers Turn Pictures into Programs," EDN Magazine Special Software Supplement, June 20, 1991, pp. 13-20.
13. Wells, G. and Baroth, E. C., "Telemetry Monitoring and Display using LabVIEW," Proceedings of National Instruments User Symposium, Austin, Texas, March 28-30, 1993.